Binding Arguments Solutions

Copyright © James Raynard 2021

Generalized predicate

- What is the following code meant to achieve? Why does it not work?

```
bool match(const string& text, const string& value) {
   return text == value;
}
cout << count_if(texts.begin(), texts.end(), match("two")) << endl;
```

  - The intention is that the algorithm calls match() with the element as the first argument and "two" as the second argument
  - It does not work because count_if's predicate can only take one argument (the element)
- Suggest two different ways it could be made to work as intended
  - std::bind
  - Lambda with capture and lambda-local variable

Copyright © James Raynard 2021

std::bind()

- Explain what std::bind() does
  - It performs a partial function call
  - std::bind() returns a callable object in which some arguments have been captured, while other arguments are left for the completion of the function call
  - "Normal" arguments (literals or variables) are captured
  - "Placeholder" arguments will be supplied when the returned object is invoked and forwarded to the original function

Copyright © James Raynard 2021

Placeholder arguments

- What syntax is used for placeholder arguments?
  - The first argument to the callable object is _1
  - The second argument to the callable object is _2
  - These are defined in the std::placeholders namespace

Copyright © James Raynard 2021

std::bind() example

- Rewrite the example from the first slide so that it works correctly, using std::bind()

Copyright © James Raynard 2021

Lambda equivalent

- Rewrite the example from the first slide so that it works correctly, using a lambda with capture and a lambda-local variable

Copyright © James Raynard 2021